SUPERIOR UNIVERSITY

Task#4

Submitted to: Sir Aqib

Submitted by: Hifza Khalid

Roll#:SU92_BSSEM_F22_202

Subject: Advance Computer Programming

Section: BSSE-4D

Date: Jan22,2024.

Topic: Double linkedlist

Question#1:

- **Double linkedlist**

CreateDoublyLinkedList.java

```
public class CreateDoublyLinkedList {

    //Create a node for doubly linked list
    class Node{
        String data;
        Node prev;
        Node next;

        public Node(String data) {
            this.data = data;
        }
```

```
    }

    //Initialize head and tail for the doubly linked list
    Node head = null;
    Node tail = null;

    //Create addNewNode() method to add a node into a list
    public void addNewNode(String data) {

        //Create node
        Node newNode = new Node(data);

        //Check whether our doubly linked list is empty or not
        if(head == null) {
            //The newNode is pointed by both head or tail
            head = newNode;
            tail = newNode;
            //It is first node so prev will point to null
            head.prev = null;
            //It is also last node so tail's next will point to null
            tail.next = null;
        }
        //Execute when the doubly linked list is not empty
        else {

            //The newly created node will be the last node, so now tail's next will point to t
hat newly created node
            tail.next = newNode;
            //The tail is pointing to the second last node so the newly created node's prev
 will point to tail
            newNode.prev = tail;
            //The newly created node will become new tail because it is last node in the d
oubly linked list
            tail = newNode;
            //The newly created node will be the last node so tail's next will be null
            tail.next = null;
        }
```

```
    }

    //Create showData() method for displaying data of doubly linked list
    public void showData() {
        //intialize a new node current that will point to head
        Node current = head;
        //Check whether the doubly linked list is empty or not
        if(head == null) {
            //Print a statement and pass the control flow into the main() method
            System.out.println("List is empty");
            return;
        }
        //Print a statement
        System.out.println("Nodes of doubly linked list: ");
        //Iterate the doubly linked list using while
        while(current != null) {
            //Print tha data on that particular node and then increment the pointer for indi
cating next node
            System.out.print(current.data + "\n");
            current = current.next;
        }
    }

    public static void main(String[] args) {

        CreateDoublyLinkedList obj = new CreateDoublyLinkedList();

        //Add nodes into the doubly linked list
        obj.addNewNode("New York");
        obj.addNewNode("Los Angeles");
        obj.addNewNode("Chicago");
        obj.addNewNode("Houston");
        obj.addNewNode("Houston");

        //Call showData() method for displaying doubly linked list data
        obj.showData();
    }
```

```
}
```

- **AddNodeInBeginning.java**

```
public class AddNodeInBeginning {

   //Creating a node for doubly linked list
   class Node{
      String data;
      Node prev;
      Node next;

      public Node(String data) {
         this.data = data;
      }
   }

   //Initializing head and tail for the doubly linked list
   Node head = null;
   Node tail = null;

   public void addNewNodeInBegin(String data) {
         //Creating node
         Node newNode = new Node(data);

         //Checking whether the list is empty or not
         if(head == null)
         {
            //The newNode is pointed by both head or tail
            head = newNode;
            tail = newNode;
            //It is first node so prev will point to null
            head.prev = null;
            //It is also last node so tail's next will point to null
            tail.next = null;
         }
         ////Execute when the list is not empty
```

```
            else {
                //The head's prev will point to the newNode
                head.prev = newNode;
                //The newNode's next will point to the head
                newNode.next = head;
                //The newNode's prev will point to null because it will be the first node
                newNode.prev = null;
                //The newNode will become new head because now the newly created nod
e is the first node of the list
                head = newNode;
            }
    }

    //Creating showData() method for displaying data of doubly linked list
    public void showData() {
        //intializing a new node current that will point to head
        Node current = head;
        //Checking whether the doubly linked list is empty or not
        if(head == null) {
            //Printing a statement and pass the control flow into the main() method
            System.out.println("List is empty");
            return;
        }
        //Printing a statement
        System.out.println("Nodes of doubly linked list: ");
        //Iterating the doubly linked list using while
        while(current != null) {
            //Print tha data on that particular node and then increment the pointer for indi
cating next node
            System.out.print(current.data + "\n");
            current = current.next;
        }
    }

    public static void main(String[] args) {

        AddNodeInBeginning obj = new AddNodeInBeginning();
```

```
        //Adding nodes into the doubly linked list
        obj.addNewNodeInBegin("Emma");
        obj.addNewNodeInBegin("Adele");
        obj.addNewNodeInBegin("Aria");
        obj.addNewNodeInBegin("Ally");
        obj.addNewNodeInBegin("Paul");


        //Calling showData() method for displaying doubly linked list data
        obj.showData();
    }
}
```

- **AddNodeAtEnd.java**

```
public class AddNodeAtEnd {

    //Creating a node for list
    class Node{
        String data;
        Node prev;
        Node next;

        public Node(String data) {
            this.data = data;
        }
    }

    //Initialize head and tail for the list
    Node head = null;
    Node tail = null;

    //Create firstNode() method for creating first node in the list
    public void firstNode(Node node){
        //The node will be pointed by both head and tail
        head = node;
```

```
        tail = node;
        //It is first node so prev will point to null
        head.prev = null;
        //It is also last node so tail's next will point to null
        tail.next = null;
    }

    //Create addNewNodeAtEnd() method to add a node at last in the list
    public void addNewNodeAtEnd(String data) {
            //Creating new node
                    Node newNode = new Node(data);

                    //Check whether the list is empty or not
                    if(head == null)
        {
            //Call firstNode() method to make it first node in the list
            firstNode(newNode);

                    }
        ////Execute when the list will not be empty
        else {
            //The newly created node will be the last node, so now tail's next will point to t
hat newly created node
            tail.next = newNode;
            //The tail will point to the second last node so the newly created node's prev
will point to tail
            newNode.prev = tail;
            //The newly created node will become new tail because it is last node in the li
st
            tail = newNode;
            //The newly created node will be the last node so tail's next will be null
            tail.next = null;
        }
    }

    //Creating showData() method for displaying data of list
    public void showData() {
```

```
        //intialize a new node current that will point to head
        Node current = head;
        //Checking whether the list is empty or not
        if(head == null) {
            //Print a statement and pass the control flow into the main() method
            System.out.println("List is empty");
            return;
        }
        //Print a statement
        System.out.println("Nodes of doubly linked list: ");
        //Iterate the list using while
        while(current != null) {
            //Print tha data on that particular node and then increment the pointer to point
 to the next node
            System.out.print(current.data + "\n");
            current = current.next;
        }
    }

    public static void main(String[] args) {

        AddNodeAtEnd obj = new AddNodeAtEnd();

        //Adding nodes at the end of the list
        obj.addNewNodeAtEnd("New York");
        obj.addNewNodeAtEnd("Chicago");
        obj.addNewNodeAtEnd("Houston");


        //Calling showData() method for displaying doubly linked list data
        obj.showData();
    }
}
```

- **AddNodeAtSpecifiedLocation.java**

```
public class AddNodeAtSpecifiedLocation {
```

```
    //Creating a node for the list
    class Node{
        String data;
        Node prev;
        Node next;

        public Node(String data) {
            this.data = data;
        }
    }
    public int size = 0;

    //Initializing head and tail for the list
    Node head = null;
    Node tail = null;

    //Create firstNode() method for creating first node in the list
    public void firstNode(Node node){
        //The node is pointed by both head or tail
        head = node;
        tail = node;
        //It is first node so prev will point to null
        head.prev = null;
        //It is also last node so tail's next will point to null
        tail.next = null;
    }

    //Create lastNode() method for adding node at last in the list
    public void lastNode(Node node){
        //The tail's next will point to that node
        tail.next = node;
        //The tail is pointing to the second last node so the newly created node's prev wi
ll point to tail
        node.prev = tail;
        //The newly created node will become new tail because it is last node in the dou
bly linked list
```

```
            tail = node;
            //The newly created node will be the last node so tail's next will be null
            tail.next = null;
        }

        //Create addNodeInBeginning() method for adding node at first position
        public void addNodeInBeginning(Node node){
            //The head's prev will point to the newNode
            head.prev = node;
            //The newNode's next will point to the head
            node.next = head;
            //The newNode's prev will point to null because it will be the first node
            node.prev = null;
            //The newNode will become new head because now the newly created node is t
he first node of the list
            head = node;
        }

        //Creating addNewNode() method to add a node into a list
        public void addNewNode(String data) {

            //Creating node
            Node newNode = new Node(data);

            //Checking whether our list is empty or not
            if(head == null) {
                //Call firstNode() method to create first node in the list
                firstNode(newNode);
            }
            //Execute when the list is not empty
            else {
                //Call lastNode() method to add a node at last in the list
                lastNode(newNode);
            }
            size++;
        }
```

```
public void addNodeToSpeciifiedPosition(int position, String data) {
    //Creating node
    Node newNode = new Node(data);

    //Checking whether our doubly linked list is empty or not
    if(head == null) {
        System.out.println("The specified location is not available");
    }
    //Execute when the list is not empty
    else {
        if(position == size+1){

            //Call lastNode() method to add the node at last in the list
            lastNode(newNode);

        }else if(position == 1){

            addNodeInBeginning(newNode);

        }else{
            //current node will point to head
            Node current = head;
            Node temp = null;

            //Iterate list till current points to the specified position
            for(int i = 1; i < position-1; i++){
                current = current.next;
            }

            //The temp node points to the node that is next to current
            temp = current.next;
            temp.prev = current;

            //newNode will be added between current and temp
            current.next = newNode;
            newNode.prev = current;
            newNode.next = temp;
```

```
                temp.prev = newNode;
            }
            size++;
        }
    }

    //Creating showData() method for displaying data of doubly linked list
    public void showData() {
        //intializing a new node current that will point to head
        Node current = head;
        //Checking whether the doubly linked list is empty or not
        if(head == null) {
            //Printing a statement and pass the control flow into the main() method
            System.out.println("List is empty");
            return;
        }
        //Printing a statement
        System.out.println("Nodes of doubly linked list: ");
        //Iterating the doubly linked list using while
        while(current != null) {
            //Print tha data on that particular node and then increment the pointer for indi
cating next node
            System.out.print(current.data + "\n");
            current = current.next;
        }
    }

    public static void main(String[] args) {

        AddNodeAtSpecifiedLocation obj = new AddNodeAtSpecifiedLocation();

        //Adding nodes into the doubly linked list
        obj.addNewNode("New York");
        obj.addNewNode("Los Angeles");
        obj.addNewNode("Chicago");
        obj.addNewNode("Houston");
        obj.addNewNode("Phoenix");
```

```
        //Adding nodes at the specified position
        obj.addNodeToSpeciifiedPosition(6, "Philadelphia");
        obj.addNodeToSpeciifiedPosition(1, "San Antonio");
        obj.addNodeToSpeciifiedPosition(3, "San Diego");

        //Calling showData() method for displaying doubly linked list data
        obj.showData();
    }
}
```

- **DeleteNodeFromList.java**

```
public class DeleteNodeFromList {

    //Create a node for DDL
    class Node{
        String data;
        Node prev;
        Node next;

        public Node(String data) {
            this.data = data;
        }
    }

    int size = 0;

    //Initialize head and tail for DDL
    Node head = null;
    Node tail = null;



    //Create addNewNode() method to add a node into DDL
    public void addNewNode(String data) {
```

```
        //Creating node
        Node newNode = new Node(data);

        //Checking whether our DDL is empty or not
        if(head == null) {
            //The newNode is pointed by both head or tail
            head = newNode;
            tail = newNode;
            //It is first node so prev will point to null
            head.prev = null;
            //It is also last node so tail's next will point to null
            tail.next = null;
        }
        //Execute when the DDL is not empty
        else {

            //The newly created node will be the last node, so now tail's next will point to t
hat newly created node
            tail.next = newNode;
            //The tail is pointing to the second last node so the newly created node's prev
 will point to tail
            newNode.prev = tail;
            //The newly created node will become new tail because it will be last node in t
he DDL
            tail = newNode;
            //The newly created node will be the last node so tail's next will be null
            tail.next = null;
        }
        //Increment the size of DDL
        size++;
    }

    //Create deleteFirstNode() method for deleting first node from DDL
    public void deleteFirstNode(){
        head = head.next;
    }
```

```
//Create deleteLastNode() method for deleting last node from DDL
public void deleteLastNode(){
    tail = tail.prev;
}

//Create deleteNodeAtSpecifiedLocation() method for deleting a node from DDL
public void deleteNodeAtSpecifiedLocation(int position){

    //Check whether the DDL is empty or not
    if(head == null || position>size) {
        System.out.println("\n" +position +"th position is not available\n");
    }
    //Execute when the DDL is not empty
    else {
        if(position == 1){

            //Call deleteFirstNode() method to delete first node from the DDL
            deleteFirstNode();

            //Decrement the size of DDL
            size--;

        }else if(position == size){
            //Call deleteLastNode() method to delete last node from the DDL
            deleteLastNode();

            //Decrement the size of DDL
            size--;

        }else{
            //current node will point to head
            Node current = head;

            //Iterate list till current points to the specified position
            for(int i = 1; i < position; i++){
                current = current.next;
```

```
            }

            //Delete node pointed by current
            current.next.prev = current.prev;
            current.prev.next = current.next;

        }
        System.out.println(position +" node is deleted successfully from DDL");
    }
}


//Create showData() method for displaying data of DDL
public void showData() {
    //intialize a new node current that will point to head
    Node current = head;
    //Check whether the DDL is empty or not
    if(head == null) {
        //Print a statement and pass the control flow into the main() method
        System.out.println("List is empty");
        return;
    }
    //Print a statement
    System.out.println("Nodes of doubly linked list: ");
    //Iterate the DDL using while
    while(current != null) {
        //Print tha data on that particular node and then increment the pointer for indi
cating next node
        System.out.print(current.data + "\n");
        current = current.next;
    }
}

public static void main(String[] args) {

    DeleteNodeFromList obj = new DeleteNodeFromList();
```

```
        //Add nodes into the DDL
        obj.addNewNode("New York");
        obj.addNewNode("Los Angeles");
        obj.addNewNode("Chicago");
        obj.addNewNode("Houston");
        obj.addNewNode("Phoenix");

        //Call showData() method for displaying DDL data
        obj.showData();

        //Delete nodes from the DDL
        obj.deleteNodeAtSpecifiedLocation(2);
        obj.deleteNodeAtSpecifiedLocation(1);
        obj.deleteNodeAtSpecifiedLocation(5);

        //Call showData() method for displaying DDL
        obj.showData();
    }
}
```